4

AD-A214 245

Centralized and Decentralized Kalman Filter Techniques for Tracking, Navigation, and Control

Chris Brown, Hugh Durrant-Whyte, John Leonard, Bobby Rao, and Barry Steer

Technical Report 277
May 1989 (revised)

UNIVERSITY OF
ROCHESTER
COMPUTER SCIENCE

89 11 01 059

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 277 (revised)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Centralized and Decentralized Kalman Filter Techniques for Tracking, Navigation, and Control		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) C.M. Brown, H. Durrant-Whyte, J. Leonard, and B. Rao		8. CONTRACT OR GRANT NUMBER(s) DACA76-85-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department 734 Computer Studies Bldg. University of Rochester, Rochester, NY 14627		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS D. Adv. Res. Proj. Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE May 1989
		13. NUMBER OF PAGES 25
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) US Army, ETL Fort Belvoir, VA 22060		15. SECURITY CLASS. (of this report) unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

none.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Navigation, Kalman Filter, Distributed Sensors, Tracking, Dynamical systems estimation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Reverse

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT

A review of some estimation basics is followed by illustrative applications of Kalman filters for stationary and maneuvering targets. The variable dimension of Kalman filter is used for the maneuvering target. The performance of the nearest neighbor standard filter is compared to that of the probabilistic data association filter for tracking a target in clutter. Multi-target tracking, using sonar sensors to estimate an autonomous robot's distance from walls, is applied to the navigation problem. The Kalman filter equations can be completely decentralized and distributed among the nodes of a multi-sensor system. Each sensing node implements its own local Kalman filter, arrives at a partial decision, and broadcasts it to every other node. Each node then assimilates this received information to arrive at its own local but optimal estimate of the system state. An appendix contains brief implementational notes.

ABSTRACT

A review of some estimation basics is followed by illustrative applications of Kalman filters for stationary and maneuvering targets. The variable dimension Kalman filter is used for the maneuvering target. The performance of the nearest neighbor standard filter is compared to that of the probabilistic data association filter for tracking a target in clutter. Multi-target tracking, using sonar sensors to estimate an autonomous robot's distance from walls, is applied to the navigation problem. The Kalman filter equations can be completely decentralized and distributed among the nodes of a multi-sensor system. Each sensing node implements its own local Kalman filter, arrives at a partial decision, and broadcasts it to every other node. Each node then assimilates this received information to arrive at its own local but optimal estimate of the system state. An appendix contains brief implementational notes.

1 INTRODUCTION AND BACKGROUND

This report includes work that sprang directly out of a reading group organized by Hugh Durrant-Whyte during the Michaelmas term of 1988. A number of projects arose, varying from exercises to long-term research projects. Much of the work described here is still in progress, and will be the subject of theses and papers.

There are some ongoing activities at the Robotics Research Group of the University of Oxford relating to optimal estimation that are independent of the reading group and which are described elsewhere.

1. Alan McIvor has worked on characterizing the properties of the extended Kalman filter when used to track edges in a dynamic environment [21].
2. Ron Daniels and his students are implementing very fast (800 Hz.) Kalman filters in his work on robot control.
3. Guy Scott and Nick Walker are actively investigating wave-propagation, especially the Schroedinger wave model, as an underlying mechanism to implement tracking and correspondence [26]. This work is described in a SKIDS paper.

The work that grew out of the reading group is presented below in the following order. Bobby Rao and John Leonard made a brief survey of estimation techniques. Barry Steer and Chris Brown implemented various examples of Kalman filters: Steer applied the filter to estimating the position of a feature, Brown implemented various tracking algorithms for single targets with and without clutter from [3], the text of the reading group. Brown implemented matrix and kalman filtering utilities used by himself and Leonard. John Leonard is working on a project to develop and implement an active sensor control strategy for one of the lab's mobile robots. The strategy is based on a hierarchical controller with different levels of Attention, Looking, and Recognition [20], using multiple sensors including vision and sonar. His contribution to this report is on multi-target tracking for localization. Bobby Rao has completed a project to define completely decentralized Kalman filters. Previous work has relied on a hierarchy of distributed sensors, but no work to date has had a completely decentralized flavor. The algorithm has been implemented on a distributed computer network, and the scope of the project is growing.

Hugh Durrant-Whyte is actively using Kalman filtering techniqes to do navigation and tracking in a sonar application. He is producing a paper on "Navigation by correlating geometric sensor data", and the authors and editor of this document owe much to his leadership and insight.

2 ESTIMATION AND KALMAN FILTERING

2.1 Estimation

2.1.1 Non-Bayesian Estimation

Non-Bayesian Estimation is used when the value being estimated is not a random variable but is *constant*. The estimate should converge to this value as the number of readings $\longrightarrow \infty$.

Assuming the prior Probability Density Function (PDF) of x is unknown, its posterior PDF is unavailable, leading to the use of a likelihood function:

$$\lambda_k(x) = p(z^k|x)$$

and hence the Maximum Likelihood (ML) method

$$\hat{x}(k) = argmax_x p(z^k|x).$$

The likelihoods arise from empirical or analytic models of the sensor.

2.1.2 Bayesian Estimation

Bayesian Estimation is used when the parameter to be found is a random variable (RV) with a PDF $p(x)$. A value of x is assumed to have occurred via this PDF and remained constant during its measurement sequence. The measurement sequence should converge to the actual value of x that we are measuring.

Given the prior PDF for x, its posterior PDF follows from Bayes' Rule

$$p(x|z^k) = \frac{p(z^k|x)p(x)}{p(z^k)}.$$

This leads to the Maximum A Posteriori (MAP) method:

$$\hat{x}(k) = argmax_x p(x|z^k) = argmax_x[p(z^k|x)p(x)].$$

Both the ML and MAP methods yield modes of a probability distribution (the most likely value).

2.1.3 Least Squares Estimation

The LS method is for non-random parameters and for measurements

$$z(j) = h(j, x) + w(j),$$

which yields

$$\hat{x}(k) = argmin_x \sum_{j=1}^{k}[z(j) - h(j, x)]^2.$$

Here $h(.)$ is the sensor model. The least squares techniques yields the mean, not the mode, of the probability distribution function.

2.1.4 Minimum Mean-Square Error Estimation

The MMSE method is for random parameters and yields

$$\hat{x}(k) = argmin_{\hat{x}} E[(\hat{x} - x)^2|z^k].$$

2.1.5 Linear Estimation

The estimator is a linear function of the measurement data.

$$\hat{x} = \bar{x} + P_{xz}P_{zz}^{-1}(z - \bar{z})$$

where

$$P_{xx} = E[(x - \bar{x})(x - \bar{x})^t],$$

$\bar{z}$ is the expected measurement, and $(z - \bar{z})$ is the *innovation* produced by the true measurement.

2.1.6 Equivalences Of Methods

All the above can produce the same results under certain conditions:

- ML $\equiv$ MAP

  This occurs when the prior PDF for x is non-informative, ie when $p(x) = \epsilon$ and $\epsilon \longrightarrow 0$ or when the PDF is a Gaussian with $\sigma_0 \longrightarrow \infty$

- LS $\equiv$ ML

  This occurs if the noises $w(j)$ in $z(j) = h(j,x) + w(j)$ are independent identically distributed (IID) zero mean Gaussian RVs. This also applies if x is a vector property.

- MMSE $\equiv$ MAP

  These coincide if the posterior PDF of x is Gaussian with arbitrary mean.

- Linear Estimation

  When x is a Gaussian RV the linear estimator of the MMSE form is equivalent to the best *linear* estimator for arbitrarily distributed RVs with the same first and second moments. However if the RVs are not Gaussian the MMSE can be a better estimator than the linear estimator (ie. Gaussian RVs give the worst case results of an MMSE estimator).

2.2 The Kalman Filter

2.2.1 Linear Kalman Filter Equations

Kalman filtering is a form of optimal estimation characterized by recursive (i.e. incremental) evaluation, an internal model of the dynamics of the system being estimated, and a dynamic weighting of incoming evidence with ongoing expectation that produces estimates of the state of the observed system. The basic Kalman filter loop appears in Fig. 1 (taken from [3]). Its input is the system measurements, its apriori information is the system dynamics and noise properties of system and measurement, and its useful outputs are the *innovation*-(the difference between the predicted and observed measurement, by which the filter's performance may be quantified), and the estimated system state.

Consider a system with state vector $\mathbf{x}$, moving under a control $\mathbf{u}$, affected by Gaussian noise $\mathbf{v}$:

$$\mathbf{x}(k+1) = \mathbf{F}(k)\mathbf{x}(k) + \mathbf{G}(k)\mathbf{u}(k) + \mathbf{v}(k). \tag{1}$$

The system makes observations $\mathbf{z}$ with Gaussian noise $\mathbf{w}$:

$$\mathbf{z}(k) = \mathbf{H}(k)\mathbf{x}(k) + \mathbf{w}(k). \tag{2}$$

where $\mathbf{v}(k)$ and $\mathbf{w}(k)$ are zero mean, white random sequences with covariances $\mathbf{Q}(k)$ and $\mathbf{R}(k)$, respectively.

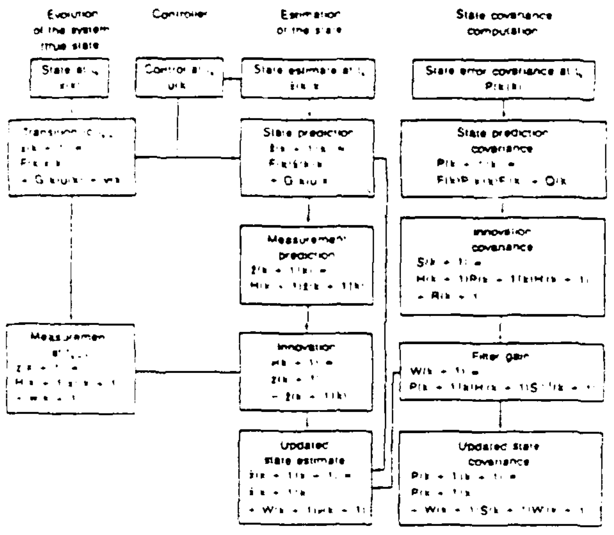


Figure 1: The (Linear) Kalman Filter (one cycle).

For such a system the conventional Kalman Filter equations for state prediction $\mathbf{x}(k+1 \mid k)$, variance prediction $\mathbf{P}(k+1 \mid k)$, state update $\mathbf{x}(k+1 \mid k+1)$ and variance update $\mathbf{P}(k+1 \mid k+1)$, where the variance is defined as

$$\mathbf{P}(k \mid k) = E\{[\mathbf{x}(k) - \hat{\mathbf{x}}(k \mid k)][\mathbf{x}(k) - \hat{\mathbf{x}}(k \mid k)]^T \mid \mathbf{Z}^k\}$$

may be found in (Bar-Shalom [3]) and have the following form:

Prediction: The state at the time of the next measurement may be predicted:

$$\hat{\mathbf{x}}(k+1|k) = \mathbf{F}(k)\hat{\mathbf{x}}(k|k) + \mathbf{G}(k)\mathbf{u}(k). \qquad (3)$$

The state prediction covariance $\mathbf{P}(k+1|k)$ is

$$\mathbf{P}(k+1|k) = \mathbf{F}(k)\mathbf{P}(k|k)\mathbf{F}^T(k) + \mathbf{Q}(k). \qquad (4)$$

The next measurement is predicted using characteristics of the measurement process and the predicted state:

$$\hat{\mathbf{z}}(k+1|k) = \mathbf{H}(k+1)\hat{\mathbf{x}}(k+1|k). \qquad (5)$$

Update: The *innovation* ν is defined as the difference between the predicted and actual next measurement:

$$\nu(k+1) = \mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k) \qquad (6)$$

The innovation covariance $S(k+1)$ is:

$$S(k+1) = H(k+1)P(k+1|k)H^T(k+1) + R(k+1). \tag{7}$$

S can be used to compute the filter gain, $W(k+1)$:

$$W(k+1) = P(k+1|k)H^T(k+1)S^{-1}(k+1). \tag{8}$$

The innovation weighted by the filter gain, plus the predicted state, form the updated state estimate:

$$\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + W(k+1)\nu(k+1). \tag{9}$$

The updated state covariance $P(k+1|k+1)$ is:

$$P(k+1|k+1) = P(k+1|k) - W(k+1)S(k+1)W^T(k+1) \tag{10}$$

2.2.2 Extended Kalman Filter Equations

The (first order) extended Kalman filter (EKF) is a version of the Kalman filter that deals with nonlinear dynamics or nonlinear measurement equations, or both. It linearizes the problem around the predicted state (a second-order EKF makes a second-order approximation). The basic control loop of Fig. 1 still applies, but measurements are predicted using the nonlinear measurement equation $h(.)$. The measurement model $h[k, x(k)]$ is linearized about the current *predicted* state vector, $\hat{x}(k+1 \mid k)$ using the Jacobian of the nonlinear measurement function $h(\cdot)$. The calculations for filter gain, state update, and covariance update use the Jacobian $h_x(k)$. Likewise state prediction is accomplished using the nonlinear state equation $f(\cdot)$. The plant model $f[k, x(k)]$ is linearized about the current *estimated* state vector, $\hat{x}(k \mid k)$. The state prediction covariance is computed using the plant Jacobian $f_x(k)$.

With these definitions of $f_x(k)$, $h_x(k+1)$ the extended Kalman filter equations are the following.

Prediction:

$$\hat{x}(k+1 \mid k) = f[k, \hat{x}(k \mid k)] \tag{11}$$

$$P(k+1 \mid k) = f_x(k)P(k \mid k){f_x}^T(k) \tag{12}$$

Update:

$$S(k+1) = h_x(k+1)P(k+1 \mid k){h_x}^T(k+1) + R(k+1) \tag{13}$$

$$K(k+1) = P(k+1 \mid k)h_x(k+1)S^{-1}(k+1) \tag{14}$$

$$\hat{\mathbf{x}}(k+1 \mid k+1) = \hat{\mathbf{x}}(k+1 \mid k) + \mathbf{K}(k+1)[\mathbf{z}(k+1) - \mathbf{h}[k, \hat{\mathbf{x}}(k+1 \mid k)]] \tag{15}$$

$$\mathbf{P}(k+1 \mid k+1) = \mathbf{P}(k+1 \mid k) - \mathbf{W}(k+1)\mathbf{S}(k+1)\mathbf{W}^T(k+1) \tag{16}$$

In the formulation above and in the examples of the next section, one process receives all measurements and makes the estimation. There has been a considerable amount of recent interest in the development of algorithms to process information obtained from a distributed sensor system. This interest arises from a need to use parallel processing architectures efficiently. Parallel computation is needed if multi-sensor systems are to be able to process their data in real-time. A decentralized filter is presented below.

3 CENTRALIZED FILTERS

3.1 Angle-only Tracking of a Point Target

As an introduction to Kalman filtering, we present an extended Kalman filter (EKF) for a system that takes angle-only measurements of a point target in motion. An example of a measurement of this kind is passive underwater sonar. This scenario arises in robotics when using infra-red range sensors, which are characterized by poor depth accuracy but which can have excellent angular resolution [11].

We consider a simplified two-dimensional system with state vector $\mathbf{x}(k) = (x_k, y_k)$ referenced to a global stationary reference frame. The target moves forward in time with known heading angle ϕ and speed T subject to additive zero-mean gaussian noise sources v_x and v_y:

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ y_k \end{bmatrix} + \begin{bmatrix} T\cos\phi \\ T\sin\phi \end{bmatrix} + \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{17}$$

This is the *plant equation*, which we can write as:

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \mathbf{u}(k) + \mathbf{v}(k), \quad \mathbf{v}(k) \sim N(\mathbf{0}, \mathbf{Q}(\mathbf{k})). \tag{18}$$

At time t_k, a sensor located at the origin takes an angular measurement $\mathbf{z}(k)$ which is the target's angular position θ_k corrupted by zero-mean gaussian noise $\mathbf{w}(k)$ with standard deviation w_θ. To relate the angle θ_k to the state vector $\mathbf{x}(k)$ we define the non-linear measurement function $\mathbf{h}[\mathbf{x}(k)]$:

$$\theta_k = \mathbf{h}[k, \mathbf{x}(k)] = \tan^{-1}\frac{y_k}{x_k}. \tag{19}$$

We can now write the *measurement equation* for this system:

$$\mathbf{z}(k) = \mathbf{h}[k, \mathbf{x}(k)] + \mathbf{w}(k), \quad \mathbf{w}(k) \sim (0, w_\theta). \tag{20}$$

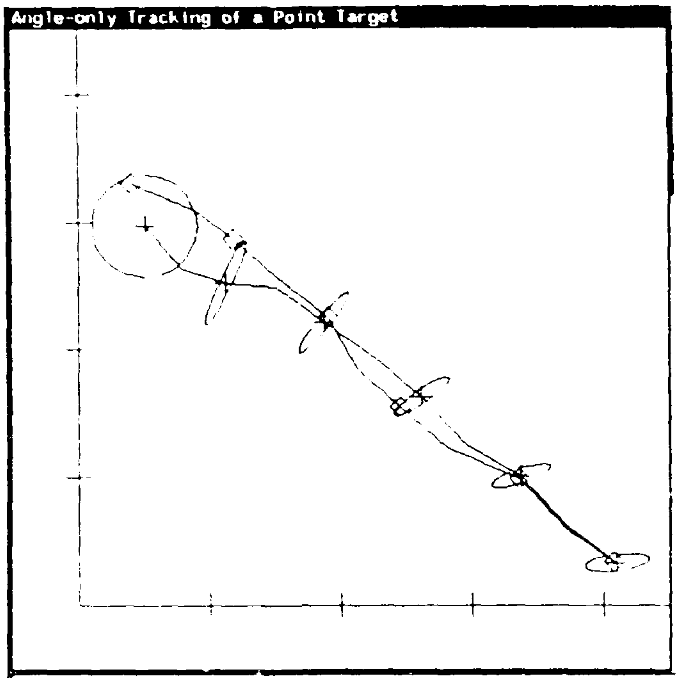


Figure 2: Tracking a point target with angle-only measurements. The target moved from left to right with heading angle $\phi = -45$ degrees and speed $T = 0.5$ meters per second. Real (diamond) and estimated (cross) target positions and *a priori* confidence ellipses are displayed every 2 seconds. The standard deviations for plant noise sources v_x and v_y are 8 cm. The measurement noise standard deviation w_θ is 1.2 degrees. The coordinate axes are marked off in 1 meter intervals.

The above plant and measurement models were implemented using the Kalman filtering facilities developed here at Oxford. Figure 2 shows a typical run, displaying the actual target position $\mathbf{x}(k)$, the *a priori* estimate $\hat{\mathbf{x}}(k \mid k)$, and an error ellipse determined by $\mathbf{P}(k \mid k)$, our confidence in this estimate. The initial error ellipse is a large circle, reflecting little confidence in the initialization. Each new angle-only measurement provides only partial information, squeezing the error ellipse in the direction perpendicular to θ_k while the error ellipse grows in the parallel direction due to plant noise.

3.2 Maneuvering Targets

When targets *maneuver*, i.e. depart from the basic, steady-state, "normal" dynamic behaviour, a tracking filter must respond. To the filter, maneuvering is signaled by a rapid increase in the normalized innovation. Recommended methods for dealing with this situation include the following.

1. Increase the process noise, or certain components of it, attributed to the target.
2. Use several filters with different assumptions in parallel, and combine their outputs probabilistically.
3. Create new filters as needed, pursuing a hypothesis tree of parallel hypotheses about target state. This tree must be pruned rapidly lest its maintenance overwhelm the computational resources.
4. Model maneuvers as colored (correlated) noise: in particular model target acceleration as a zero mean, first-order Markov process (one with exponential autocorrelation).
5. Perform *input estimation*, in which measurements based on the nonmaneuvering model are used to detect and estimate the control input applied to the plant dynamics, and that control input in turn is used to correct the state estimate.
6. Use *variable dimension filtering*, in which the maneuver is considered part of the plant dynamics, not noise. Maneuver detection causes the substitution of a different, higher-order dynamic model for the lower-order, "quiescent" model.

Bar Shalom [3] compares the performance on maneuvering targets of three filters: two level white noise, variable dimension (VD) filtering, and input estimation (IE). He notes that the computational effort ratios between the three are 1:2:8. His study reveals that the two level white noise filter does surprisingly well, being slightly worse than the VD filter but definitely better than the IE filter.

We chose to implement the VD filter, in the light of its relatively low computational cost and relatively high efficacy in the Bar-Shalom study. Our illustrative application was to a target moving in two dimensions at constant velocity until some time at which it begins constant acceleration in the same direction. The quiescent filter is simply the constant-velocity target filter, the maneuvering filter is for a constant acceleration target.

Fig. 3(a) shows the normalized innovation (i.e. error measure) of the constant-velocity filter as time passes. Performance starts degrading at $T = 5$, when acceleration begins. Fig. 3(b) shows the corresponding estimate of y position, which gradually degrades through time. The VD filter has remarkably better performance, which underlines the importance of accurate dynamic modeling. Fig. 3(c) shows the normalized innovation of the VD filter (note the scale change). The maneuvering filter was switched in at $T = 6$. Fig. 3(d) shows the corresponding estimate of y position.

3.3 Targets in Noise

Tracking an object in the presence of spurious measurements (*clutter*) can be done in several ways. All assume a *validation gate* outside of which measurements are ignored: its size is a function of the desired probability of including the true measurement, and can be derived from a chi-squared calculation applied to the normalized innovation.

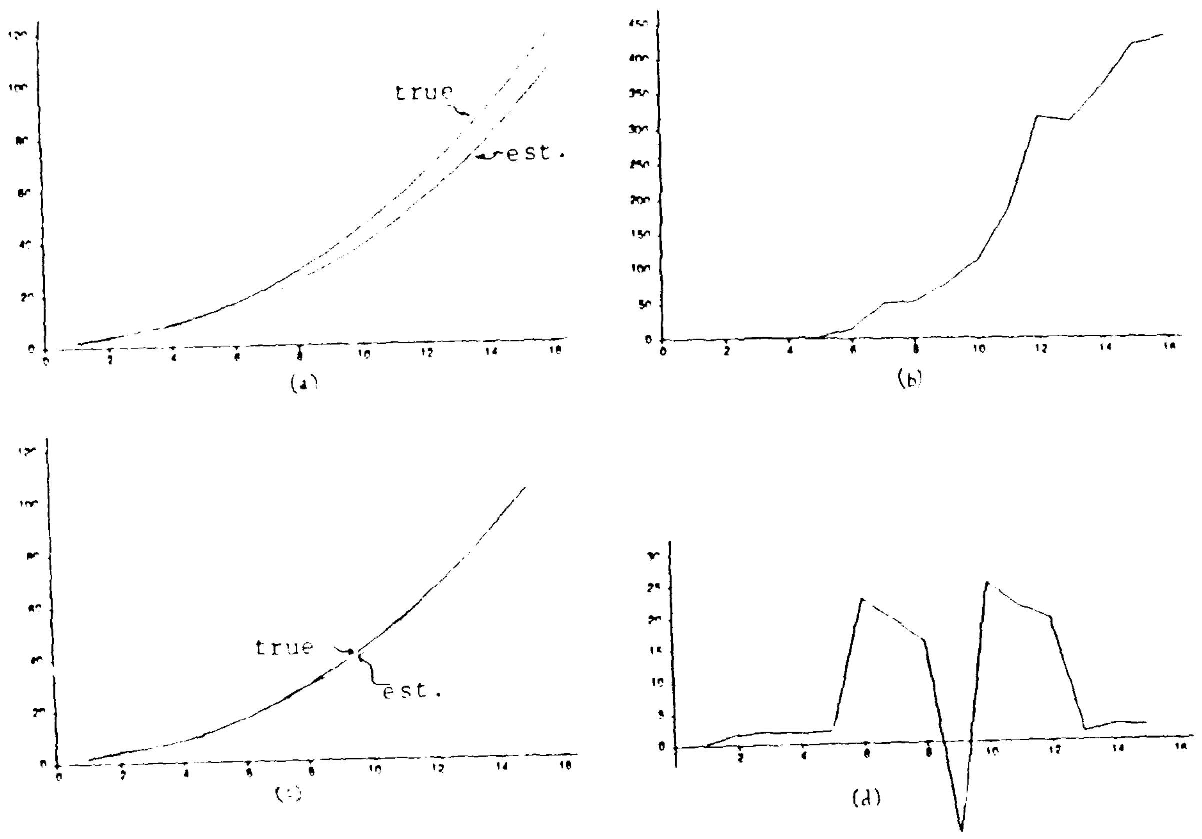
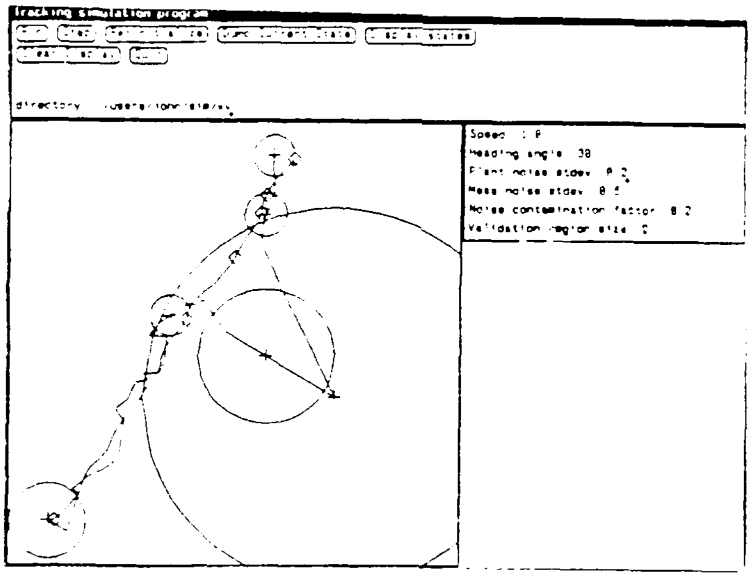


Figure 3: Performance of the Variable Dimension filter. (a) Estimated y position for target that starts *maneuvering at* $T=5$. (b) Normalized innovation of quiescent filter applied to target. (c) *Estimated* y *position* for target from VD filter. (d) Normalized innovation of VD filter (note scale change compared to (b)).



```
Speed : 1.0
Heading angle 30
Plant noise stdev 0.2
Meas noise stdev 0.5
Noise contamination factor 0.2
Validation region size 2
```

Figure 4: Real (diamond) and estimated (cross) target tracks, and elliptical validation gate, through time, showing loss and reacquisition of track.

1. The optimal way to track a single target in clutter is the *track-splitting* approach, in which a tree of possible tracks is maintained. This can be a combinatorially expensive method.

2. An obvious alternative to treating all measurements, as it were, in parallel, is to pick a single candidate measurement and *proceed as if it were the right one.* The obvious candidate is the one closest in measurement space (that one with smallest normalized innovation), so this technique is called the *nearest-neighbor standard filter* (NNSF). The problem is that the true measurement can be missed.

3. A third approach is the *probabilistic data association filter*, or PDAF. In it, the measurements within the validation gate are probabilistically blended to yield a combined innovation which is input into the Kalman filtering process. The problem is that the result does not correspond to that of any actual measurement.

The application illustrated in Figure 4 is tracking a constant-velocity target moving in two dimensions. The plant and measurement are both noisy, with the measurement noise being drawn from a contaminated Gaussian, in which with some probability the measurement noise has different parameters (here, higher variance) than the noise expected by the filter. The filter produces a validation gate, based on the innovation covariance. A measurement within the validation gate is used, while one outside the gate is ignored. In the run illustrated, the elliptical validation gate (here a circle) shrinks while good measurements are obtained, and grows when a measurement is missed — this adjustment makes reacquisition more likely. The figure shows snapshots of the validation gate during a serious loss and reacquisition of track.

General aspects of the behaviour of the NNSF and PDAF filters may be predictable on abstract grounds. For instance we might make the following predictions for uniformly distributed clutter and high *probability of detection* (probability that the target is detected at all, either inside or outside the validation gate.)

1. With both the NNSF and PDAF filters tracking in clutter, as time goes on it is increasingly likely that the filter will "lose track", e.g. start tracking clutter, or have an estimate of target state outside some fixed bound.
2. With a "non-maneuvering" NNSF filter, low and high clutter levels may be less immediately harmful than medium levels, since with low clutter the target is likely to be nearest the predicted state, and with high clutter there is likely to be a clutter point near the predicted state. It would seem that at intermediate level the clutter would be more likely to attract the filter away from the target.
3. The NNSF filter would seem more likely to make serious errors by tracking clutter since it does not weigh the evidence. The performance of the PDAF should degrade more gracefully as conditions get worse.

We implemented the NNSF and PDAF filters, and used them to provide individual output tracks, as in the previous work. Also the programs were embedded in Monte Carlo simulations to provide data over a number of runs in statistically similar situations. The results confirm the above expectations but also provide some surprises. Fig. 5 shows individual tracking runs for situations with uniformly distributed clutter of increasing density. The number of clutter points in the validation gate was determined by rounding a normal variate of indicated mean (the clutter density) and standard deviation to the nearest integer, and uniformly distributing the resulting number of clutter points throughout the validation gate volume. In these runs the volume was close to unity. The NNSF figures should be compared with Fig. 6, which shows PDAF results for similar situations.

Figs. 5 and 6 illustrate indeed that lower clutter levels can result in worse NNSF performance than higher levels, and that performance of both filters falls off as time increases. They perhaps furnish a mild surprise, *viz.* the ability of the PDAF to reacquire the track. This happens for lower clutter levels, and presumably occurs when the "signal to noise" ratio is high in the validation gate (e.g. there are few measurements in the validation gate, and at least one of them is near the actual position of the target and correctly is accorded high weight). The behavior of the PDAF in high clutter conditions is not as surprising — it drifts, taking the average of the random clutter.

Fig. 7 shows statistics gathered over $N = 50$ runs with the NNSF filter. It should be compared to Fig. 8. The plots show the fraction of *lost tracks* and the *average final error* of the filter's estimate. Both functions vary over the set of tracking times {4, 8, 16, 32 } timesteps, and both vary over the set of clutter densities { 0.25, 0.5, 1.0, 1.5, 2.0, 3.0 }. Here "lost track" is defined as the estimated position being more than some fixed distance (here 2.0) from the actual position of the target. Thus it is possible to imagine the filter actually tracking clutter for the entire run, going wildly wrong in its estimates, but luckily

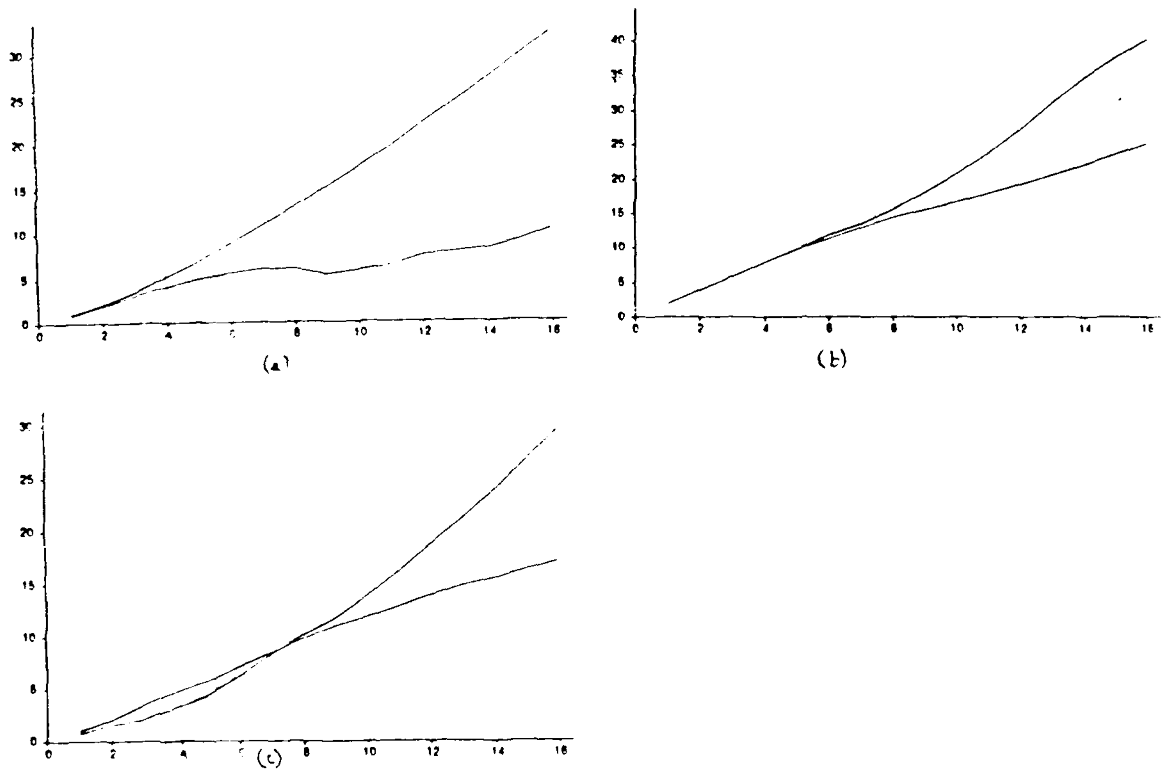


Figure 5: In this figure and the next, the filter is a constant-velocity (linear) Kalman filter. The plant noise is an acceleration component (white noise of mean 0.0 and variance $q = 0.2$.) Clutter density $\sigma = 0.4$. (Parallelepidal) validation gate size is such that .99 of target measurements should fall within it initially. The measurement noise has variance 0.2 for the x, 0.1 for y. (a) Error in X vs time in NNSF, tracking situation parameters as in text, clutter density 0.5. (b) Error in Y vs time in NNSF, clutter density 1.0. (c) Error in X vs time in NNSF, clutter density 2.0.

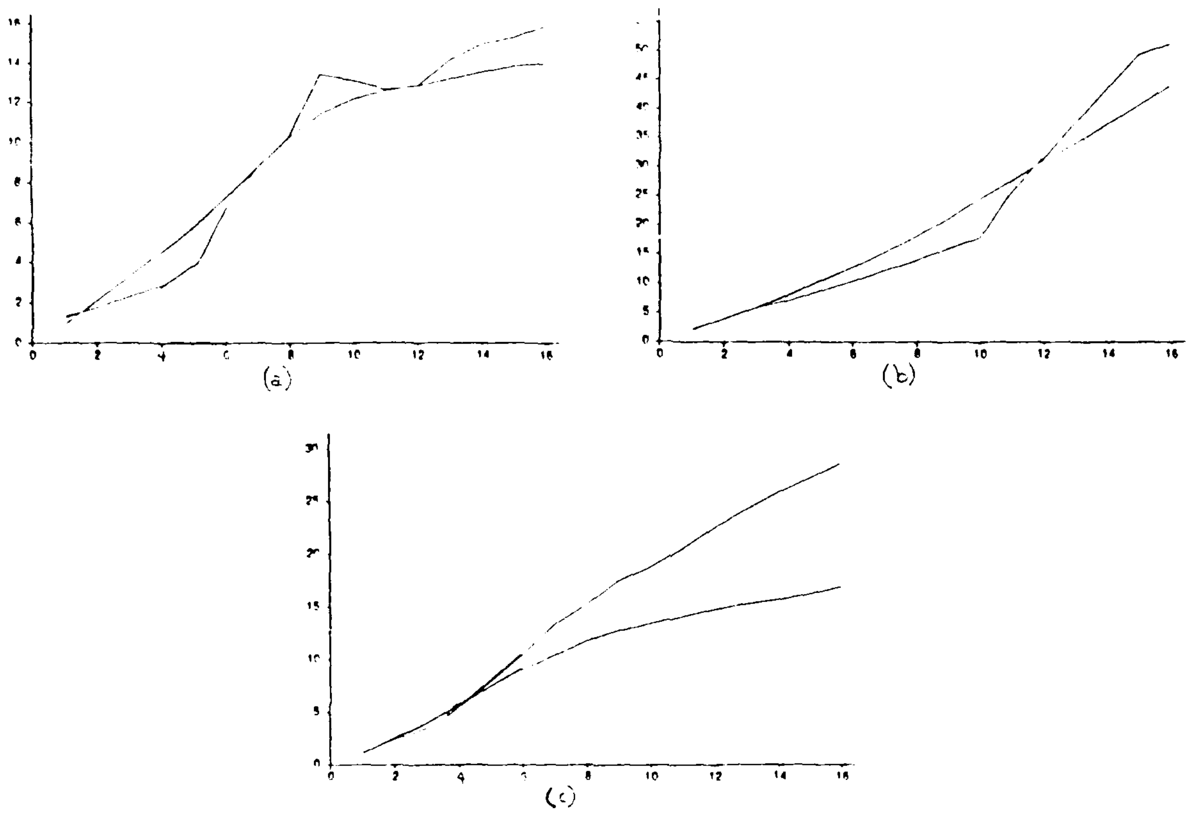


Figure 6: (a) Error in X vs time in PDAF, tracking situation parameters as in text, clutter density 0.5. (b) Error in Y vs time in PDAF, clutter density 1.0. (c) Error in X vs time in PDAF, clutter density 2.0.

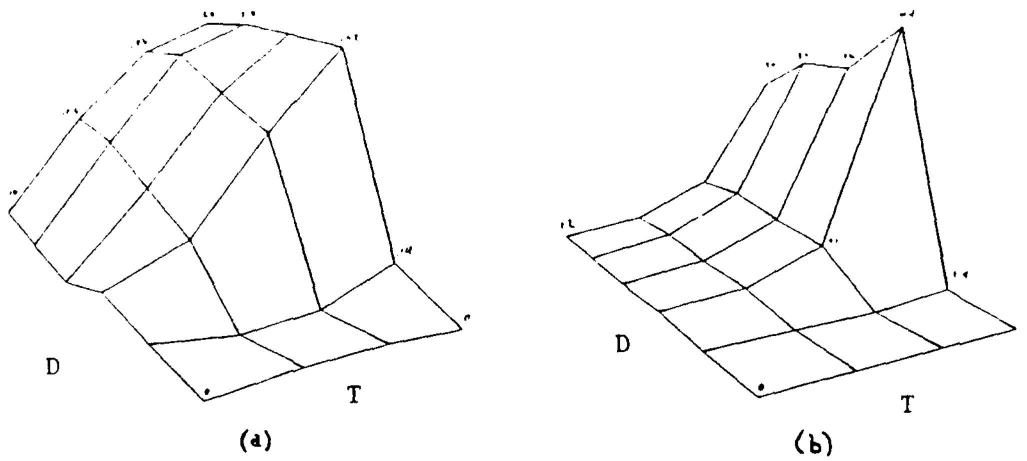


Figure 7: This figure and the next show the fraction of *lost tracks* (a) and the *average final error* (b) of the filter's estimate. Both functions vary over the set of tracking times {4, 8, 16, 32 } timesteps (axis T), and both vary over the set of clutter densities { 0.25, 0.5, 1.0, 1.5, 2.0, 3.0 } (axis D). "Lost track" means the estimated position is more than some fixed distance from the actual position of the target. This figure shows results for the NNSF, with tracking situation as in previous figures.

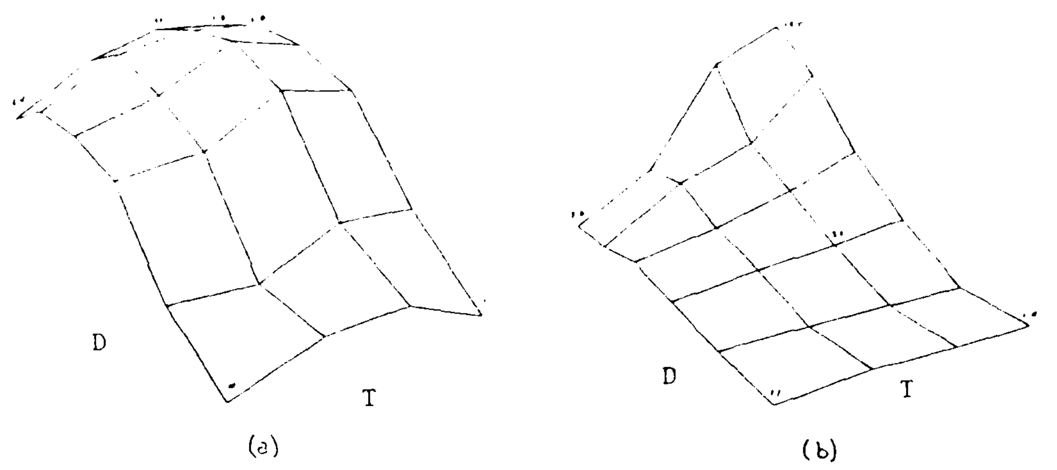


Figure 8: Lost tracks (b) and average final error (b) vs track length and clutter density PDAF, with tracking situation as in last figure.

arriving inside the threshold distance just at the last step, and thus not "losing track" by this definition.

The average final error function is meant to quantify the filter performance more than the discrete "lost track" measure, illustrating a linear loss function corresponding to the intuition that an estimate closer to the truth at the final timestep is better. Results like those in Figures 5 and 6, we are probably safe in inferring that a closer final estimate also betokens a better estimate throughout the run.

The plots are perhaps surprising in that by this definition of lost track, the NNSF outperforms the PDAF fairly convincingly over a large range. These results are typical of many we obtained, but it is occasionally possible to engineer situations where the PDAF loses track less often. At least it seems fair to say that the situation is more complicated than it appears from Figure 6.1 of [3], which indicates a marked superiority of PDAF along axes whose semantics are not clear from the text (perhaps the original paper gives more details).

The plots are perhaps not surprising in that they accord with the prediction of graceful degradation of the PDAF in terms of the average error metric, using which it convincingly outperforms the NNSF. The higher sensitivity of NNSF to intermediate clutter levels is again demonstrated.

3.4 Navigation by Multi-target Tracking

We are using a Kalman-filter-based, multi-target tracking approach to mobile robot navigation. The goal is to get optimal performance with a single sensor through a strategy of *Active Sensor Control.* Ultimately, we shall use a hierarchical control strategy with three layers (*attention, looking, recognition*). In the limited navigational task we undertake first (maneuvering in a known laboratory with unknown obstacles), the recognition aspect is subsumed into the looking aspect.

The navigation problem can be divided into the two distinct tasks of localization (determining the absolute position of the robot), and obstacle avoidance. In the multi-target tracking framework, localization is a process of *looking* for *expected* events; the sensors are directed to instantiate the locations of beacons, which are reliable, easy-to-sense, natural features of the robot's environment, such as the walls of the room. Precise localization of position can then be obtained by using a target-based [3] approach to track these beacons as the robot moves. The task of obstacle avoidance is achieved by a measurement-based [25] tracking technique whose attention is captured by unexplained events in the path of the robot. We show the localization component of this navigation technique using a single sonar sensor mounted on a simple mobile robot with point dynamics. Although the scheme has been run on the robot hardware, the illustrations come from a simulator we are developing that will incorporate detailed sensor and environment models for sonar.

Ultimately this work will incorporate both the "top-down" and "bottom-up" control styles that can be found in perception (see the Discussion section). In the full implementation, the *recognition* layer is complex, involving path-planning, obstacle avoidance, and model learning and maintenance — slow processes of a global nature. The attention layer provides a high-speed interface to the real world.

3.4.1 The System Model

The system dynamics for one of Oxford's mobile vehicles are illustrated in Fig. 9(a). The state of the vehicle at time t_k is described by the state vector $\mathbf{x}(k) = (x_k, y_k, \phi_k, T_k)$. x_k and y_k are the x and y coordinates of the center of the vehicle referenced to a global frame. ϕ_k is the orientation of the vehicle referenced to a global frame. T_k is the speed of the vehicle at time t_k, which represents the linear distance the vehicle will travel during the time interval from t_k to t_{k+1}. The motion of vehicle is such that during each timestep it travels first in a straight line forward from position (x_k, y_k) to position (x_{k+1}, y_{k+1}), then rotates to its final orientation ϕ_{k+1}.

$$\begin{aligned} x_{k+1} &= x_k + T_k \cos\phi_k \\ y_{k+1} &= y_k + T_k \sin\phi_k \end{aligned}$$

The robot motion is controlled by specifying a control input $\mathbf{u}(k) = (0, 0, \Delta\phi_k, \Delta T_k)^T$. $\Delta\phi_k$ controls the final rotation applied after the linear movement forward for the current time-step. This rotation is subject to a random disturbance v_ϕ, modeled as zero-mean Gaussian with variance q_ϕ:

$$\phi_{k+1} = \phi_k + \Delta\phi_k + v_\phi, \quad v_\phi \sim N(0, q_\phi)$$

ΔT_k controls the distance traveled forward for the *next* time-step, which is subject to a random disturbance v_T, modeled as zero-mean Gaussian with variance q_T:

$$T_{k+1} = \Delta T_k + v_T, \quad v_T \sim N(0, q_T)$$

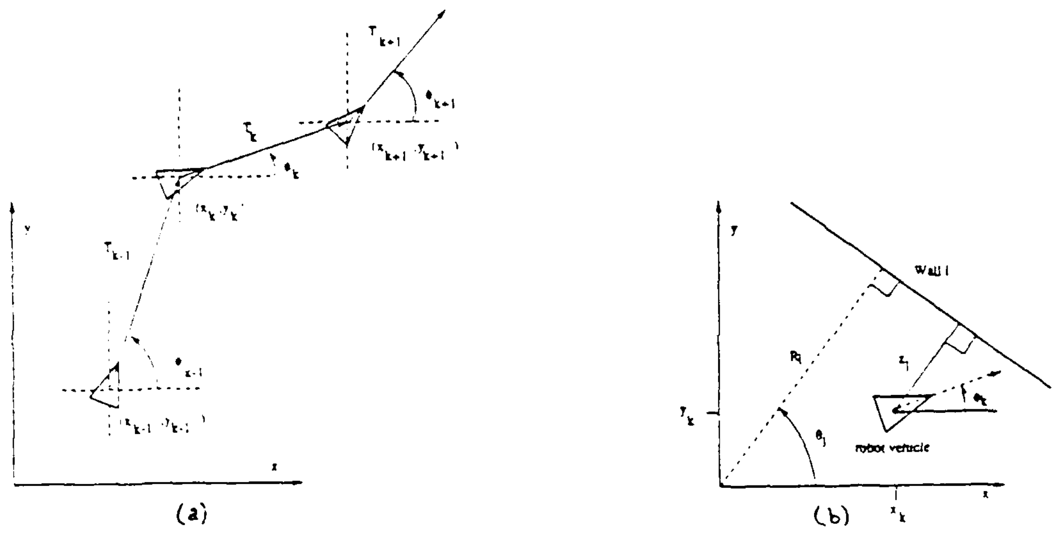


Figure 9: (a) System dynamics for robot vehicle. (b) Taking a range measurement from a wall (x_k, y_k, ϕ_k) is the position of the robot vehicle in global coordinates. (R_i, θ_i) represents a line in global coordinates. z_i is a noisy measurement of the perpendicular distance from the vehicle to wall p_i.

The overall plant equation of the system is

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \phi_{k+1} \\ T_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + T_k \cos\phi_k \\ y_k + T_k \sin\phi_k \\ \phi_k \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \Delta\phi_k \\ \Delta T_k \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ v_\phi \\ v_T \end{bmatrix} \tag{21}$$

which may be written as:

$$\mathbf{x}(k+1) = \mathbf{f}[k, \mathbf{x}(k)] + \mathbf{u}(k) + \mathbf{v}(k), \quad \mathbf{v}(k) \sim N(\mathbf{0}, \mathbf{Q}(\mathbf{k})) \tag{22}$$

where $\mathbf{f}[k, \mathbf{x}(k)]$ is a non-linear function of the system state.

3.4.2 The Measurement Model

The measurement model is for a stationary robot vehicle taking range measurements to a number of walls using a sonar sensor. For now, we assume perfect measurements of ϕ_k, the orientation of the vehicle. (Our system has a 9-bit resolution digital compass that gives us this information. We also have developed reliable algorithms to extract orientation information directly from the sonar data). The model of the environment is a list of n line segments that represent walls and large planar objects in the room. A line segment $\mathbf{p}_i$ is represented by R_i, its perpendicular distance from the origin, and θ_i, its orientation with respect to the origin. In terms of these parameters, the equation of the line is:

$$R_i - x\cos\theta_i - y\sin\theta_i = 0 \tag{23}$$

The sensor returns the minimum distance of any object detected in its 30 degree beam width. Thus if it is pointed in a direction approximately perpendicular to the wall, the range value obtained is the perpendicular distance from the vehicle to the wall. The active sensor control framework uses an *a priori* estimate of position $\hat{\mathbf{x}}(k \mid k-1)$ to direct the sensor to look nearly perpendicular to a given wall $\mathbf{p}_i = (R_i, \theta_i)$ to obtain a measurement z_i of the perpendicular distance to the wall (Fig. 9(b)).

Let r_i denote the true perpendicular distance from a given vehicle location (x_k, y_k) to the wall. Then

$$r_i = |R_i - x_k \cos\theta_i - y_k \sin\theta_i|. \tag{24}$$

The absolute value arises since the vehicle could be on either side of the wall. The measurement z_i is the true perpendicular distance r_i corrupted by zero-mean Gaussian noise w with variance w_r:

$$z_i = r_i + w, \quad w \sim N(0, w_r)$$

Suppose that at time t_k the robot is predicted to be in the location $\hat{\mathbf{x}}(k \mid k-1)$. The predicted location is used to suggest n walls to obtain range readings from. The sensor is directed to look for each of these walls, and the returned values are checked with an appropriate validation gate to yield validated range readings z_i for m of the n visible walls. We combine the validated range measurements to form a composite measurement vector $\mathbf{z}(k) = [x_1, \ldots, x_m]^T$. To relate this measurement vector $\mathbf{z}(k)$ to the vehicle position $\mathbf{x}(k)$, we define the non-linear function $\mathbf{h}$

$$\mathbf{h}[k, \mathbf{x}(k)] = \begin{bmatrix} |R_1 - x_k \cos\theta_1 - y_k \sin\theta_1| \\ \cdot \\ \cdot \\ \cdot \\ |R_m - x_k \cos\theta_m - y_k \sin\theta_m| \end{bmatrix} \tag{25}$$

Finally, a composite noise vector $\mathbf{w}(k)$ completes the measurement model:

$$\mathbf{z}(k) = \mathbf{h}[k, \mathbf{x}(k)] + \mathbf{w}(k), \quad \mathbf{w}(k) \sim N(\mathbf{0}, \mathbf{R}(k)) \tag{26}$$

The function $\mathbf{h}[k, \mathbf{x}(k)]$ incorporates the prior information concerning the walls $\mathbf{p}_i$ for which validated range readings have been obtained at this time interval. So far we have ignored the data association problem, assuming that each range measurement comes from the appropriate wall. The viability of this assumption depends on how accurately the system model predicts the vehicle's movement; experimentally it has been justified so far.

3.4.3 Extended Kalman Filter Equations

The extended Kalman filter equations for the plant and measurement models described above in equations 22 and 26 use linearized plant and measurement models. The plant model $\mathbf{f}[k, \mathbf{x}(k)]$ is linearized about the current *estimated* state vector, $\hat{\mathbf{x}}(k \mid k)$. We accomplish this by defining the plant Jacobian, $\mathbf{f}_{\mathbf{x}}(k)$:

$$\mathbf{f}_{\mathbf{x}}(k) = [\nabla_{\mathbf{x}} \mathbf{f}^T(k, \mathbf{x})]^T_{\mathbf{x}=\hat{\mathbf{x}}(k|k)} = \begin{bmatrix} 1 & 0 & -T_k \sin\phi_k & \cos\phi_k \\ 0 & 1 & T_k \cos\phi_k & \sin\phi_k \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}_{\mathbf{x}=\hat{\mathbf{x}}(k|k)} \tag{27}$$

The measurement model $\mathbf{h}[k, \mathbf{x}(k)]$ is linearized about the current *predicted* state vector, $\hat{\mathbf{x}}(k+1 \mid k)$. We define the measurement Jacobian $\mathbf{h}_{\mathbf{x}}(k+1)$ to be:

$$\mathbf{h}_{\mathbf{x}}(k+1) = [\nabla_{\mathbf{x}} \mathbf{h}^T(k, \mathbf{x})]^T_{\mathbf{x}=\hat{\mathbf{x}}(k+1|k)} = \begin{bmatrix} \pm\cos\theta_1 & \pm\sin\theta_1 & 0 & 0 \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \pm\cos\theta_m & \pm\sin\theta_m & 0 & 0 \end{bmatrix} \tag{28}$$

The $\pm$ signs arise because $\mathbf{h}[k, \mathbf{x}(k)]$ contains the absolute value function for reasons given above. In calculating $\mathbf{f}_{\mathbf{x}}(k)$, the appropriate sign for each wall is chosen based on the dead-reckoning estimate of the robot's position. The absolute value functions would make the Jacobian of $\mathbf{h}[k, \mathbf{x}(k)]$ unstable at sensing locations very close to the wall. This is not a problem in practice because the sensor is located at the center of the vehicle. The definitions of $\mathbf{f}_{\mathbf{x}}(k)$ and $\mathbf{h}_{\mathbf{x}}(k+1)$ are used in the extended Kalman filter equations (11) - (16) in a multi-target control structure.

3.4.4 Results

We avoid some of the filter initialization difficulties by starting the vehicle from an exactly-known initial location and setting the initial state covariance matrix $\mathbf{P}(0 \mid 0)$ to zero. Because the motion of the robot vehicle has a significant unpredictable component, the state covariance matrix $\mathbf{P}(k \mid k)$ grows considerably with time if no updates of position from the sensor are provided.

Four runs of the simulated system are shown in Fig. 10. The error ellipse is defined by the first four elements of $\mathbf{P}(k \mid k)$, and represents the uncertainty in the prediction of the vehicle position (x_k, y_k). In Fig. 10(a), the speed noise v_T causes the uncertainty ellipse to grow with time in the direction of motion of the vehicle. The heading angle noise v_ϕ causes the uncertainty ellipse to grow in the direction perpendicular to the vehicle's motion. Our task for the measurement process is to decrease the size of this ellipse, and hence increase the confidence in the state estimate $\hat{\mathbf{x}}(k \mid k)$. Fig. 10(b) and (c) show the effects of measurements from one and two walls, respectively. Fig. 10(d) shows a run in a complex environment with many walls. The sensor is directed to take measurements from all visible walls.

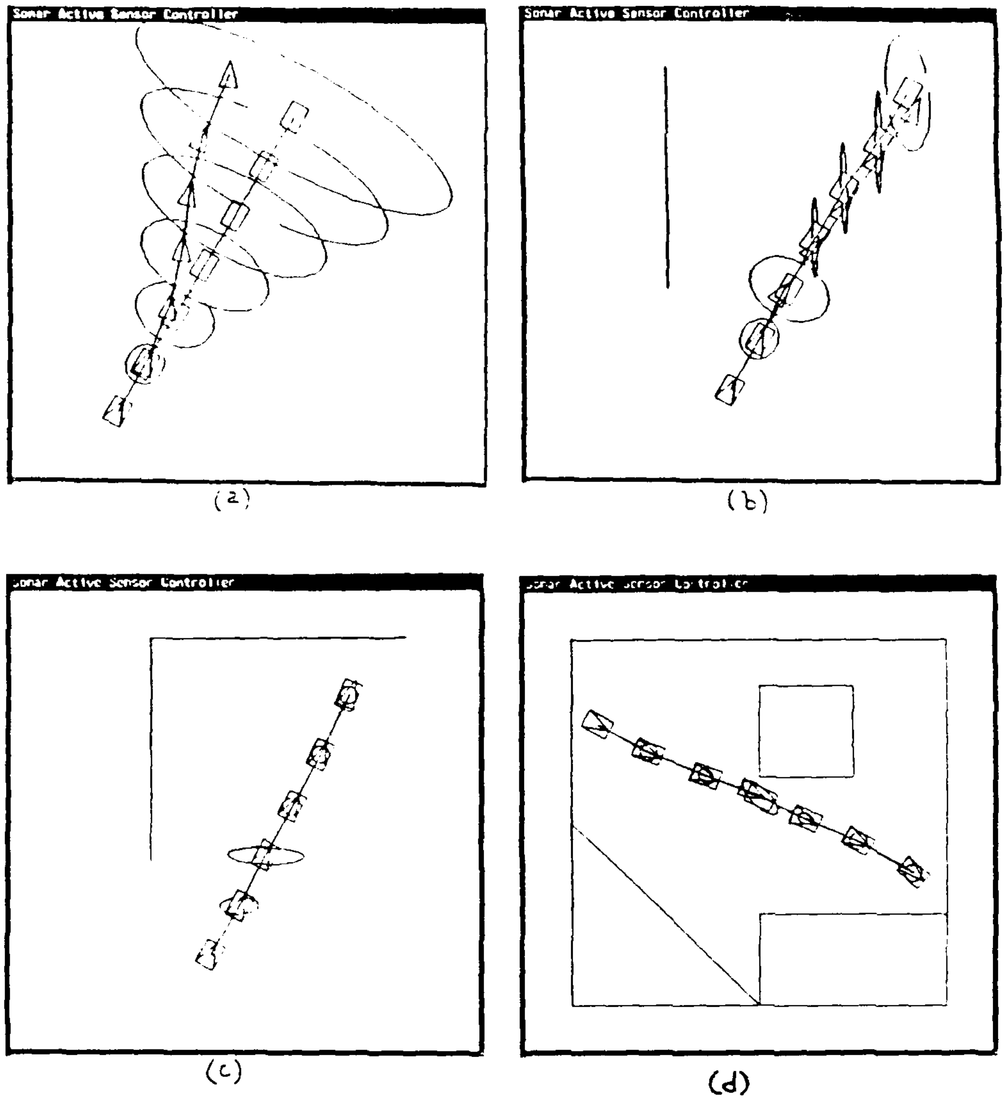

Sonar Active Sensor Controller
(a)
Sonar Active Sensor Controller
(b)
Sonar Active Sensor Controller
(c)
Sonar Active Sensor Controller
(d)

Figure 10: (a) A run with no validated measurements. The triangle represents the actual vehicle position and orientation (x_k, y_k, ϕ_k), the rectangle represents the estimated vehicle position and orientation, and the ellipse represents the uncertainty in the estimates of x_k and y_k. (b) A run taking range measurements from a single wall. The error ellipse shrinks perpendicular to the wall as *a posteriori* confidence in the estimate of x_k and y_k increases with measurements. The wall comes into view and disappears from view during the run, causing large effects in the error ellipse. (c) Measurements from two walls. Information from two directions reduces error. (d) A run in a complex environment using measurements from all walls visible from a given location.

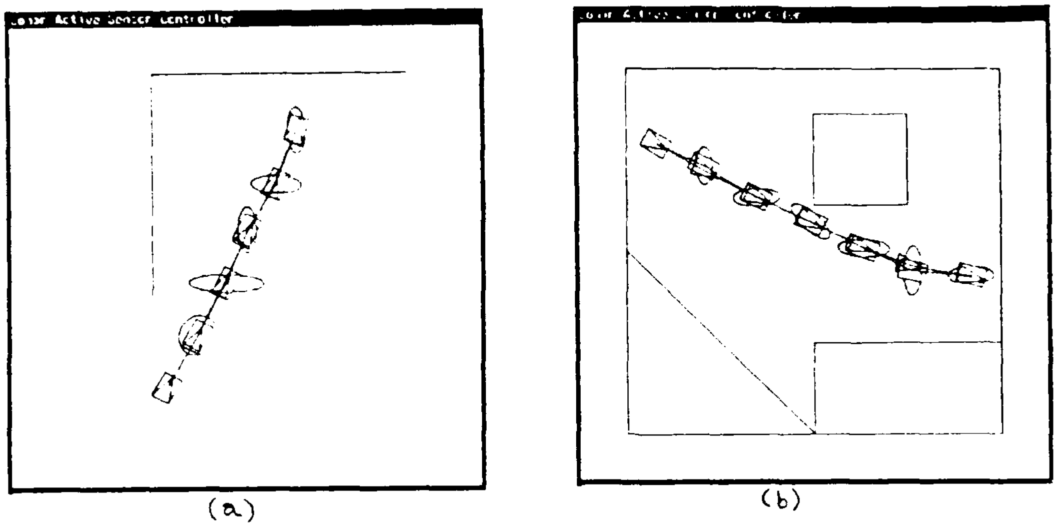


Figure 11: Choosing the best Sensing target for (a) simplified two-wall environment and (b) multiple-wall environment. All visible sensing targets are evaluated to choose the target that provides the most useful information based on the *a priori* estimate of x_k and y_k. The sensor then *looks* for this "best" wall to update the vehicle position.

3.4.5 Choosing the Best Sensing Target

Since range measurements from a single wall provide only partial information, a wall-based sonar localization algorithm needs to take measurements from several different directions, as illustrated in the examples above. However, it is undesirable in many cases to make the vehicle stop to take observations of several different walls from the same location. To give the vehicle a dynamic, "on-the-fly" localization capability we advocate a strategy of *controlling* the sensor to track different walls successively as the vehicle moves.

Our task is to keep the state estimate covariance $\mathbf{P}(k \mid k)$ as small as possible. This corresponds to keeping the error ellipse determined by $\mathbf{P}(k \mid k)$ from growing too large in any one direction. Using this criterion, the best sensing target is the wall most nearly perpendicular to the largest eigenvector of $\mathbf{P}(k \mid k)$. Fig. 11 shows runs in two-wall and multiple-wall environments that follow this control policy of choosing the best wall. This strategy provides a means of localizing position "on-the-fly" with a single sensor by focusing attention on different walls that come in and out of view as the robot moves.

3.4.6 Future Work

In this implementation, using a single Kalman filter to derive the position of the robot is effective because

1. The environment has been known exactly *a priori*
2. The environment has no moving objects and no clutter.

In a practical situation, an ultrasonic navigation system may need to deal with an unknown environment that contains unmodeled walls, unknown moving objects, and clutter. For this reason, in addition to the central Kalman filter for vehicle position, a multi-target framework that initializes and maintains tracks for individual targets in the environment is necessary. We plan to implement the three-layer strategy of Active Sensor Control as a mechanism for managing the multi-target tracking approach to navigation. In the sensor control framework, we shall develop a set of local attention processes, each tracking a particular target or searching for new targets, and carrying out some fast, local processing to facilitate this local control. The looking layer will perform the multi-target tracking *per se*, assigning local tracking resources to follow individual pre-identified targets and responding to new targets and unexplained events.

4 A DECENTRALIZED FILTER

4.1 Fully Decentralized Decision Making

Previous attempts at decentralized Kalman filtering have either assumed a completely centralized architecture (Fig. 1) in which all the sensors' observations are passed back to a central processing facility that fuses the data, [3], [8], [7], or assumed *some* level of local embedded processing capability in each sensing node, but still however relying on a central processing facility to perform the global data fusion (ie. hierarchical decentralization) [16], [17]. Since both of these methods require a single central facility to perform overall data fusion they suffer from the associated problems: potential computational bottlenecks and the susceptibility to total system failure if the central facility should fail. Harris and White [15] give descriptions of algorithms for decentralized system architectures using blackboards for communication between nodes, but they suffer from the well-known problems associated with blackboard systems (such as keeping the blackboard current and free of redundant information).

Fully decentralized decision making [13] is advantageous to hierarchically decentralized and other forms of distributed system architectures because:

1. It allows complete parallelization of any algorithm. Therefore the system would be ideal for implementation on a parallel processing network (such as a transputer array).

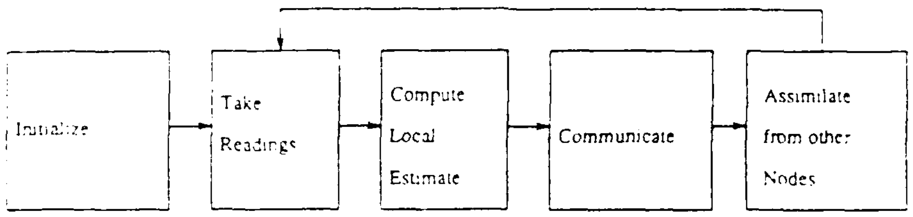


Figure 12: Algorithm for individual nodes.

2. It leads to a speed increase over other architectures by removing potential computational bottlenecks.
3. It gives a system that is very resilient to loss of one or more of its nodes (ie. a highly *survivable* system).

Distributed sensing and algorithms for sensor fusion that can be distributed amongst several sensing nodes have interesting applications in robotics, process plants and C^3I fields. The algorithm we present here is fully decentralized, requiring no central processing facility to perform data fusion [24]. The information communicated between nodes is simple and the equations for assimilation of other nodes' data are no more complex than those for a conventional Kalman Filter. Our filter reaches the same global optimum as a conventional Kalman Filter, but since this filter may be run simultaneously at the m sensing nodes of an m node system it performs faster.

4.1.1 The Decentralizing Algorithm

Fig. 12 shows the operations performed by each one of the nodes in the system. Each node is initialized with estimates of the state of the system together with an associated variance of that estimate. These initial values for state estimates and variances may be obtained in several ways and the methods used by the implementations given in this paper are outlined in later sections. After initialization, the main loop begins with each node taking a reading from its sensor of the state of the system. With this reading (and its associated variance) the node is able to compute conventional Kalman Filter equations to reach its own, new estimate of the state. Each node then broadcasts this estimate to the other nodes and receives information being broadcast to it. Last, each node computes assimilation equations to take into account the data it has just received; in fact, each node thus locally computes a global estimate.

This paper includes details of a linear version of the algorithm in which each node shares the same coordinate frame and has the same representation of the state of the system as other nodes. In the nonlinear case, each node, although measuring the same state, is in its own coordinate frame and uses its own representation of the state which may or may not coincide with the representation used by other nodes. This case is treated in full in [24] and is not derived here, although experimental results are given.

4.1.2 The Linear Algorithm

Consider a system with state vector **x**, taking observations **z** with Gaussian noise sources **w** and **v** respectively:

$$\mathbf{x}(k+1) = \mathbf{F}(k)\mathbf{x}(k) + \mathbf{G}(k)\mathbf{w}(k) \tag{29}$$

$$\mathbf{z}(k) = \mathbf{H}(k)\mathbf{x}(k) + \mathbf{v}(k) \tag{30}$$

where

$$E\left\{ \begin{pmatrix} \mathbf{w}(k) \\ \mathbf{v}(k) \end{pmatrix} \begin{pmatrix} \mathbf{w}^T(j), \mathbf{v}^T(j) \end{pmatrix} \right\} = \begin{pmatrix} \mathbf{Q}(k) & \mathbf{C}(k) \\ \mathbf{C}^T(k) & \mathbf{R}(k) \end{pmatrix} \delta_{k,j} \tag{31}$$

Often it is true that **C** (the covariance between plant and measurement noise) is zero.

For such a system the conventional Kalman Filter equations for state prediction $\mathbf{x}(k+1 \mid k)$, variance prediction $\mathbf{P}(k+1 \mid k)$, state update $\mathbf{x}(k+1 \mid k+1)$ and variance update $\mathbf{P}(k+1 \mid k+1)$, where the variance is defined as

$$\mathbf{P}(k \mid k) = E\{[\mathbf{x}(k) - \hat{\mathbf{x}}(k \mid k)][\mathbf{x}(k) - \hat{\mathbf{x}}(k \mid k)]^T \mid \mathbf{Z}^k\}$$

may be found in (Bar-Shalom [3]) and have the following form:

Prediction:

$$\hat{\mathbf{x}}(k+1 \mid k) = \bar{\mathbf{F}}\hat{\mathbf{x}}(k \mid k) + \mathbf{GCR}^{-1}\mathbf{z}(k) \tag{32}$$

$$\mathbf{P}(k+1 \mid k) = \bar{\mathbf{F}}\mathbf{P}(k \mid k)\bar{\mathbf{F}}^T + \mathbf{GQG}^T \tag{33}$$

Update:

$$\hat{\mathbf{x}}(k+1 \mid k+1) = [\mathbf{I} - \mathbf{WH}]\hat{\mathbf{x}}(k+1 \mid k) + \mathbf{Wz}(k+1) \tag{34}$$

$$\mathbf{P}^{-1}(k+1 \mid k+1) = \mathbf{P}^{-1}(k+1 \mid k) + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H} \tag{35}$$

where

$$\mathbf{W} = \mathbf{P}(k+1 \mid k+1)\mathbf{H}^T\mathbf{R}^{-1} \tag{36}$$

$$\bar{F} = F - GCR^{-1}H \tag{37}$$

To decentralize them we must make the following **Assumption** (from Hashemipour [16]): Assume we can partition the observation vector into m subvectors of dimension m_i. Assume also that we can partion the $\mathbf{H}$ matrix conformably. (This means a node cannot make a full-rank observation of the system state.) We can therefore say that

$$\mathbf{v}_i(k) = [\mathbf{v}_i^T(k), ... \mathbf{v}_m^T(k)]^T$$

and that

$$E\{\mathbf{v}(k)\mathbf{v}^T(k)\} = blockdiag\{\mathbf{R}_i(k), ..., \mathbf{R}_m(k)\}$$
$$\mathbf{H}(k) = \{\mathbf{H}_i^T(k), ..., \mathbf{H}_m^T(k)\}$$
$$\mathbf{z}(k) = \{\mathbf{z}_i^T(k), ..., \mathbf{z}_m^T(k)\}$$

From this assumption we can also partition the **F** and **G** matrices and also the state estimates and variances, $\hat{\mathbf{x}}$ and **P**. This allows each node to implement its own local Kalman Filter for its own local estimate of the state.

Each node, i has a system model and takes observations that are individualized versions of eq. (29) - (37). That is, simply subscript the following variables in those equations with i: **x**, **z**, **F**, $\bar{\mathbf{F}}$, **G**, **H**, **Q**, **C**, **R**, **P**, **W**.

4.1.3 Derivation of Assimilation Equations

We now derive the equations each node computes to assimilate the information supplied by other nodes. References to equations (29) - (37) mean their local versions. From the **Assumption** above,

$$\mathbf{H}^T\mathbf{R}^{-1}(k)\mathbf{H} = \sum_{i=1}^{m}[\mathbf{H}_i^T(\mathbf{R}_i^{-1}(k)\mathbf{H}_i] \tag{38}$$

and also

$$\mathbf{W}(k)\mathbf{z}(k) = \mathbf{P}(k \mid k)\sum_{i=1}^{m}[\mathbf{H}_i^T\mathbf{R}_i^{-1}(k)\mathbf{z}_i(k)]. \tag{39}$$

From (35) we can say

$$\mathbf{H}_i^T\mathbf{R}_i^{-1}\mathbf{H}_i = \mathbf{P}_i^{-1}(k+1 \mid k+1) - \mathbf{P}_i^{-1}(k+1 \mid k). \tag{40}$$

Hence from (35), (38) and (40) we can write

$$\mathbf{P}^{-1}(k+1 \mid k+1) = \mathbf{P}^{-1}(k+1 \mid k) + \sum_{j=1}^{m}[\mathbf{P}_j^{-1}(k+1 \mid k+1) - \mathbf{P}_j^{-1}(k+1 \mid k)].$$

and by placing this assimilation equation at each node (decentralizing) we arrive at

$$\mathbf{P}_i^{-1}(k+1 \mid k+1) = \mathbf{P}_i^{-1}(k+1 \mid k) + \sum_{j=1}^{m}[\mathbf{P}_j^{-1}(k+1 \mid k+1) - \mathbf{P}_j^{-1}(k+1 \mid k)].$$

The summation over j does not include the term when $j = i$ since this has already been accounted for in (35).

Now premultiplying (40) by $\mathbf{P}_i(k+1 \mid k+1)$ and rearranging gives

$$\mathbf{I} - \mathbf{W}_i\mathbf{H}_i = \mathbf{P}_i(k+1 \mid k+1)\mathbf{P}_i^{-1}(k+1 \mid k). \qquad (41)$$

Premultiplying (34) by $\mathbf{P}_i^{-1}(k+1 \mid k+1)$, using (41) and rearranging gives

$$\mathbf{H}_i^T\mathbf{R}_i^{-1}\mathbf{z}_i(k+1) = \mathbf{P}_i^{-1}(k+1 \mid k+1)\mathbf{x}_i(k+1 \mid k+1) - \mathbf{P}_i^{-1}(k+1 \mid k)\mathbf{x}_i(k+1 \mid k). \quad (42)$$

Using (34), (41) and (42) and decentralizing gives

$$\begin{aligned}\hat{\mathbf{x}}_i(k+1 \mid k+1) &= \mathbf{P}_i(k+1 \mid k+1)[\mathbf{P}_i^{-1}(k+1 \mid k)\hat{\mathbf{x}}_i(k+1 \mid k)\\ &+ \sum_{j=1}^{m}\{\mathbf{P}_j^{-1}(k+1 \mid k+1)\hat{\mathbf{x}}_j(k+1 \mid k+1) - \mathbf{P}_j^{-1}(k+1 \mid k)\hat{\mathbf{x}}_j(k+1 \mid k)\}\;].\end{aligned}$$

From (32) we can write

$$\mathbf{G}_i\mathbf{C}_i\mathbf{R}_i^{-1}\mathbf{z}_i(k) = \mathbf{x}_i(k+1 \mid k) - \bar{\mathbf{F}}_i\mathbf{x}_i(k \mid k) \qquad (43)$$

and noting that from the **Assumption** we can write

$$\mathbf{GCR}^{-1}\mathbf{z} = \sum_{i=1}^{m}[\mathbf{G}_i\mathbf{C}_i\mathbf{R}_i^{-1}\mathbf{z}_i]$$

we can derive

$$\hat{\mathbf{x}}_i(k+1 \mid k) = \bar{\mathbf{F}}_i(k)\hat{\mathbf{x}}_i(k \mid k) + \sum_{j=1}^{m}\{\hat{\mathbf{x}}_j(k+1 \mid k) - \bar{\mathbf{F}}_j(k)\hat{\mathbf{x}}_j(k+1 \mid k+1)\}.$$

4.1.4 Assimilation Equations

To reiterate: each node takes readings from its sensor, computes its local version of equations 33 - 34, communicates to all other nodes (see below) and then computes the equations given below.

Prediction:

$$\hat{\mathbf{x}}_i(k+1 \mid k) = \bar{\mathbf{F}}_i(k)\hat{\mathbf{x}}_i(k \mid k) + \sum_{j=1}^{m}\{\hat{\mathbf{x}}_j(k+1 \mid k) - \bar{\mathbf{F}}_j(k)\hat{\mathbf{x}}_j(k+1 \mid k+1)\} \qquad (44)$$

$$\mathbf{P}_i(k+1 \mid k) = \bar{\mathbf{F}}_i(k)\mathbf{P}_i(k+1 \mid k+1)\bar{\mathbf{F}}_i^T(k) + \mathbf{G}_i(k)\bar{\mathbf{Q}}_i(k)\mathbf{G}_i^T(k) \tag{45}$$

Update:

$$\hat{\mathbf{x}}_i(k+1 \mid k+1) = \mathbf{P}_i(k+1 \mid k+1)[\mathbf{P}_i^{-1}(k+1 \mid k)\hat{\mathbf{x}}_i(k+1 \mid k) + \sum_{j=1}^{m} \underbrace{\{\mathbf{P}_j^{-1}(k+1 \mid k+1)\hat{\mathbf{x}}_j(k+1 \mid k+1) - \mathbf{P}_j^{-1}(k+1 \mid k)\hat{\mathbf{x}}_j(k+1 \mid k)\}}_{state\ error\ info}] \tag{46}$$

$$\mathbf{P}_i^{-1}(k+1 \mid k+1) = \mathbf{P}_i^{-1}(k+1 \mid k) + \sum_{j=1}^{m} \underbrace{\{\mathbf{P}_j^{-1}(k+1 \mid k+1) - \mathbf{P}_j^{-1}(k+1 \mid k)\}}_{variance\ error\ info} \tag{47}$$

using $\hat{\mathbf{x}}(0 \mid -1) = \hat{\mathbf{x}}_0$ and $\mathbf{P}(0 \mid -1) = \mathbf{P}_0$ for initialization of each node.
(The summations above are over every node j from $j = 1$ to $j = m$ except for when $j = i$.)
The terms *state error info* and *variance error info* are the values transmitted by each node to each other node during the communication step of Fig. 12. The information to be communicated is not complex (one vector and one matrix per iteration), and the data fusion equations are no more complex than the local estimate update equations.

4.2 Implementation

The algorithms have been implemented on a SUN 3.0 computer simulating several nodes running simultaneously. The application is a pursuer-evader game in which there are a number of independent pursuers attempting to surround and trap a moving evader. The pursuers and evader move on a two dimensional surface: the evader only moves horizontally and with a constant velocity. Each pursuer is an independent mobile sensing node taking noisy observations of the single moving evader. Each pursuer has its own model of the system (ie. the evader's motion pattern) and they all communicate between themselves to arrive at a global decision of the location of the evader so that they can then decide where best to move to surround it (see Fig. 13).

4.2.1 Linear Case

In this case each pursuer takes full-state $\mathbf{z} = (x, y, \dot{x})^T$ observations of the evader, corrupted by noise (the noisiness of a node's readings being proportional to the distance between that pursuer and the evader) and each has a correct model of the evader's dynamics. The evader is travelling in the horizontal direction at constant velocity. After taking readings the pursuers evaluate local versions of equations 33 - 34 and then calculate the data they must transmit all the other pursuers. Each communicates this data to each other pursuer

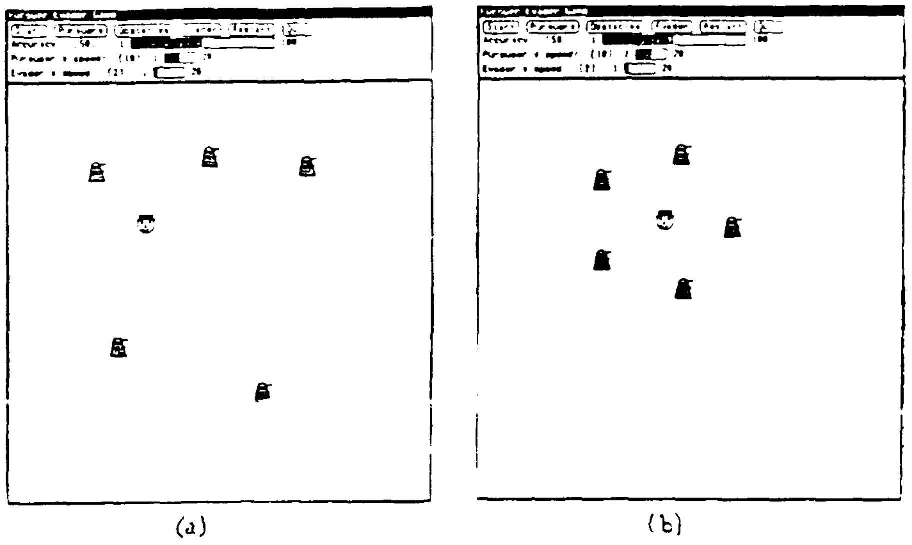


Figure 13: (a) A typical starting configuration of pursuers and the evader. (b) The pursuers have surrounded the evader.

and collect others' *variance error info* and *state error info* to arrive at a global estimate for the (x, y) position of the evader and also its velocity in the x direction. The Kalman filter is initialized by setting the observed velocity of the evader to zero for the first five iterations to allow the filter to settle before the noisy observations of the velocity are taken into account.

4.2.2 Results

Figs. 14 and 15 are graphs of the global estimates from two pursuers (only two are shown for clarity) and the evader's actual parameters plotted against number of iterations. These results show:

1. Each node arrives at the <u>same</u> (albeit initially inaccurate) global estimate from the very first iteration. Therefore, by communication the evaders have reached a common group consensus of the position of the evader.
2. The global estimates for the (x, y) positions soon converge to very close to the actual values and continue to track it well. The filter converges faster than a conventional Kalman Filter since it is able to process m lots of sensors' information simultaneously.
3. The global estimate for the velocity of the evader $\dot{x}$ converges more slowly to the actual value but this is to be expected since $\dot{x}$ is small compared to the observation noise and also differentiation always amplifies noise.

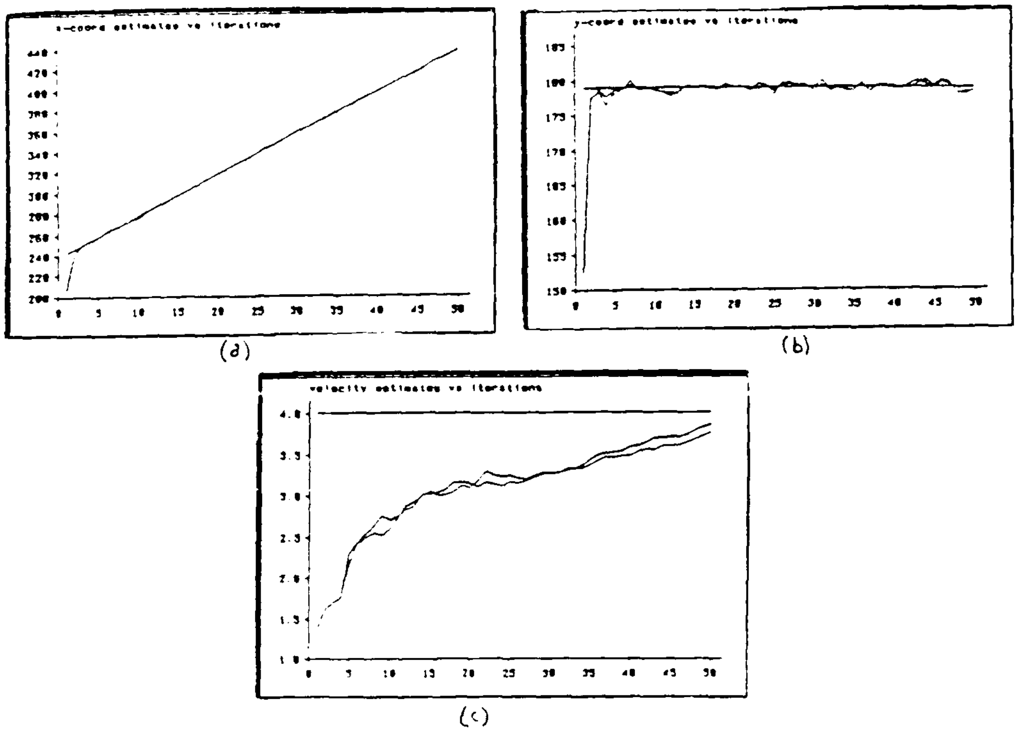


Figure 14: (a) Linear case x-coordinate estimate (actual = solid rising line). (b) Linear case y-coordinate estimate (actual = 178). (c) Linear case horizontal velocity estimate (actual = 4.0).

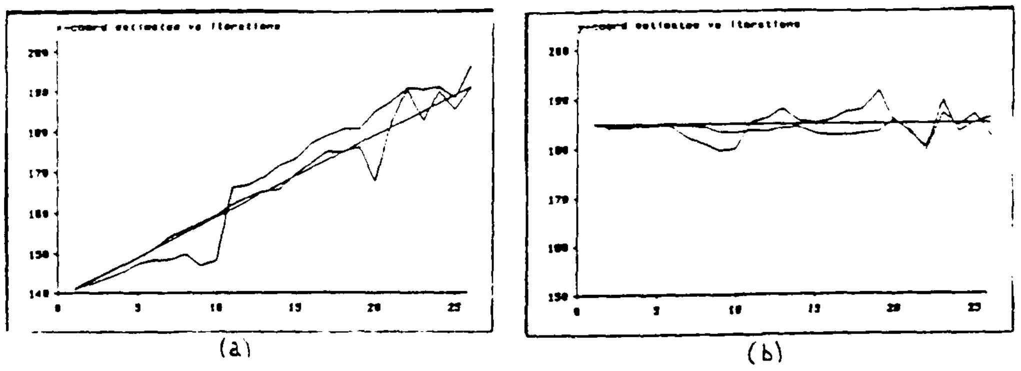


Figure 15: (a) Non-linear case x-coordinate estimate (actual = solid rising line). (b) Non-linear case y-coordinate estimate (actual = 185).

4.2.3 Nonlinear Case

In this case each pursuer has an $(r, \theta, \dot{r}, \dot{\theta})$ state representation of the position of the evader, which is moving in the horizontal direction with a constant velocity. However, in order to simulate a realistic case where each pursuer is a photodiode based device, *angle only measurements are taken.* No range data is available from the sensors so communication between the pursuer is essential for pursuers to arrive at a full state vector. Each pursuer takes its sensor measurement and constructs its own view of the state of the object. Each pursuer knows where it is and where all the other pursuers were from the last communication step so each pursuer is able to transmit appropriately transformed variance error and state error information to the others. (Also each pursuer transmits its current position to all the others). The assimilation equations are computed and each pursuer ends up with a full state representation of the evader. This 'filling in' of the empty states occurs because if one computes the linear weighted sum of a node's state estimate $\mathbf{x}_i$ and another node's transformed estimate ${}^i\mathbf{x}_j$ the new state vector $\mathbf{x}_i$ obtained is full rank (ie. the intersection of the angle estimates of the two nodes has been solved to give range data) [9]. If

$$\mathbf{x}_i = ({}_i\Sigma_T \mathbf{x}_i + {}^j_i\Sigma_T {}^i\mathbf{x}_j)({}_i\Sigma_T + {}^j_i\Sigma_T)^{-1}$$

where $\mathbf{x}_i$ and $\mathbf{x}_j$ are (r, θ) state vectors with initially no values for r and ${}_i\Sigma_T$ and ${}_j\Sigma_T$ are the associated information matrices with zeroes in the range information positions one obtains the correct estimate for r in $\mathbf{x}_i$. The filter required five steps to arrive at reasonably accurate estimates for the r and $\dot{r}$ terms for each pursuer. Then it was able to follow the evader, updating its estimates for $\dot{r}$ and $\dot{\theta}$ on each iteration.

4.2.4 Results

Results are shown in Fig. 15, which shows the (x, y) position of the evader and the (x, y) estimates of two pursuers. Only two pursuers' estimates are shown for clarity. The (r, θ) state representations of each pursuer have been converted to cartesian form for these plots, simply so that the estimates of two pursuers may be directly compared. The results show that:

1. The pursuers are able to track the evader despite each pursuer taking only a partial estimate of the state. By communication between themselves they are able to arrive at a full state vector estimate for the position of the evader.
2. The pursuers are in agreement over the position of the evader. The non-linearities in the system (ie. using an (r, θ) representation for a linear cartesian motion) cause the slight error between the two pursuers' estimates.

4.3 The Future

The algorithm has successfully been implemented in OCCAM on an array of sensors, each with a photodiode based angle detector and its own transputer. The algorithm is being tested on real data, in experiments with sensors returning measurements asynchronously. Also the algorithm may be extended to allow tracking of several objects simultaneously.

The motivation behind this work was to attempt to discover and set down mathematically what pieces of information must be communicated between a group of sensors in order for them to reach a conclusion. This problem must now be investigated further and beyond the limits of just a simple linear tracking filter algorithm. It appears that the multi-sensor network we have analysed here may be thought of as a complex Markov chain with information propagating through each node. This line of research borders onto the wider issue of sensor models and may throw some light on how best to model complex sensors such as CCD cameras in order that the value of the information returned by a sensor in any arbitrary situation may be assessed by other sensors. Being able to model sensors in this way would lead to the possibility of building intelligent sensing nodes that could be connected together as a fully decentralized network.

5 DISCUSSION

5.1 Optimal Estimation and Active Intelligence

Optimal estimation techniques have at least three distinct roles to play in real-time sensorimotor systems.

1. They can be used as the basic paradigm for estimating the state of systems *internal* to the observer. Estimating external states is akin to *perception*.

2. They can be used to estimate the state of systems *internal* to the observer. Estimating internal state involves aspects of *proprioception* (using information from internal sensors), but can also involve sensing the outside world, especially to determine dynamic observer parameters such as location and velocity.

3. They can be used as low-level utilities in service of several aspects of perception or action.

5.1.1 Control Styles in Perception

Top-down (expectation-driven, hypothesis-verification) methods cope with the inherent underdetermined and computationally intensive nature of perception by using apriori knowledge to constrain the space of interpretations for perceptual data. In a navigation context, these methods correspond to map-guided route-finding, perhaps landmark recognition and similar tasks in which an internal model exists and the input is expected.

One important role of perception is to cope with the unexpected. This seeming truism is often ignored, and has deep implications for computational perceptual models. In particular it implies that tops-down control strategies are by themselves inadequate. In a navigational context, obstacle avoidance illustrates this role. The complementary control strategy is *bottom-up*, or data-driven: Here the style is often a fixed order of processing of input data (say by successive levels of feature detection and extraction) leading to increasingly abstract levels of description of the input. As technology improves it is becoming possible to achieve the massive data-processing effort in real time, and the practical considerations that have partially motivated the tops-down approach are vanishing (see, e.g. [6]).

In one sense, the Kalman filter is an example of expectation-driven perception. By definition it incorporates explicit models of dynamics and noise. The strength of the Kalman filter for estimation is that it has these models at its disposal, but requiring them limits the sorts of perceptual jobs that the Kalman filter can reasonably be expected to perform. The severe tops-down requirements can be mitigated to some extent, and at some cost, by such measures as running several different filters embodying different assumptions in parallel, switching between filters when lack of fit motivates such a switch, allowing the filter to estimate control inputs to the plant, etc., as we have seen in earlier sections.

Despite such seemingly sophisticated adaptive capabilities, the extensive literature on Kalman filtering applications (e.g. [3,25,1,2,12,14,18,22,23,10] reveals that the *perceptual* tasks most often attempted are those in which the plant (often *target*) follows well-known and rather simple (e.g. ballistic) dynamic laws, and in which the target is modeled as a point in space. The typical perceptual task is tracking the (perhaps maneuvering) target (perhaps in clutter). Thus the perceptual task (*data association*, or *segmentation*), consists of the twofold problem of linking measurements together into tracks and ignoring spurious data. The basic Kalman filter mechanism provides help in the way of quantified measures of *uncertainty*, *surprise*, *information*, *expectation*, etc. but provides nothing directly to cope with the familiar problems of controlling search in interpretation space. The track-splitting, NNSF, PDAF, etc. approaches all have analogs in the edge-linking problem in computer vision, for example.

Dissatisfaction with the paradigm of expectation-driven low-level perception, combined with skepticism about the efficacy of local bottom-up segmentation, has motivated other algorithms for target tracking. Their goal is often to accomplish perceptual grouping using global, expectation-driven metrics of grouping quality, as well as to cope with input in a more data-driven way [19]. However, much remains to be done here.

5.1.2 Perception: Estimating External State

It seems that if optimal estimation techniques are to be a paradigm for *perception*, at least the following conditions must apply.

1. The dynamics of the objects must obey known, predictable laws.
2. The noise mean and covariance properties of the domain dynamics and of the measurement system must be known apriori.

3. The data may arise from several information sources — the Kalman filtering technique provides a principled way to combine (fuse) them.

4. The raw perceptual input must be processed to yield a measurement vector containing information about the state of the observed objects. If objects are more complicated than single points, this step may call for solving "the vision problem" in order to do tracking. An extreme example is reliable tracking of a face in a crowd, using data from a face-recognizer. The point is that even such basic vision tasks as region-finding are not well understood, and should not be lightly suggested as "preprocessing" for a tracker.

5. Measurement data must be available over a significant interval, probably tens of time-steps for reasonably complex domain dynamics.

6. Dealing with complex perceptual events in real time will call for substantial computational resources.

5.1.3 Proprioception: Estimating Internal State

The other main use for Kalman filters is for internal state estimation, in aid of complex, often adaptive, control (e.g., [10]). Such *proprioception* is not divorced from perception: A vehicle can determine its position from fixed beacons (say landmarks or stars) by tracking them and interpreting the data with a Kalman filter. In practice, the contrast between the sophistication of the dynamic models for plants that are to be estimated and controlled and the models of external targets is dramatic (for a plant to be controlled, sometimes 80 state variables, for a target to be tracked, perhaps six). Presumably the observer's state description equations are relatively stable, and devoting computational and analytic resources to precise observer description is a good long-term investment that will pay off in better information about and control over its state. For another thing, the observables themselves are under more control. A roving vehicle can observe bar-coded reflectors as location beacons, or can track static features such as walls or furniture, whose apparent motion arises more or less predictably from observer motion.

Thus the proprioception problem is inherently more "expectation driven" than the perception problem, and Kalman filtering techniques may well be an appropriate paradigm since the following conditions apply.

1. The dynamics of the observer obeys known, predictable laws. They may be complex to model but there is only a single, known system to characterise, and it makes sense that the observer is something the observer itself knows best.

2. Proprioception may be provided by on-board sensors such as tachometers, odometers, shaft encoders, etc.

3. The noise mean and covariance properties of the observer dynamics and its measurement system can be experimentally determined "off-line", as can the properties of the expected visual stimuli.

4. The raw perceptual input must be processed to yield a measurement vector containing information about the state of the observer, but the observer can choose to interpret a limited set of stimuli by customized methods if internal state estimation is the only goal.

5. Presumably measurement data is available for a significant interval, on the order of the "lifetime" of the observer, rather than of the lifetime of the unexpected visual phenomena that occur in perception.

6. Computational requirements may not be as severe since the update rate needed for proprioception may be lower than that for perception.

5.2 The Future

We plan to use both aspects of estimation in work at Rochester to integrate real-time vision and motion with high-level planning in a hierarchical parallel system. An early step has been the inclusion of estimation techniques in the robot's gaze control system [4,5]. This work is in the simulation stage, but a simulator is needed in any event to implement the Smith predictor used to cope with software delays in the system. The simulator uses a suboptimal, time-invariant filter (the $\alpha - \beta$ filter [3]) on the two-dimensional image data and the three-dimensional target location data presumed available from binocular stereo, kinetic depth calculations, or other means.

6 APPENDIX

6.1 The Kalman Filter Utilities

The EKF utilities are supported by a standard suite of matrix routines. The filter, plant and observer are implemented as structures. There is very little use of global storage: instead there is storage attached to structures, and static working storage within routines. Filters may currently be connected in linked lists, and (should the need arise) the linking mechanism is easy to extend to support trees and forests of filters. The code is in several files, some of which should not need to change with the application, some of which must. They are available in cmb/src/ekf. The PDAF application (the most recent) is in .../pdaf. also needed are various header files in cmb/include and the statistical and matrix utilities in cmb/lib.

6.1.1 Sample Main Loop

The main control loop for the example of the variable dimension filter looks approximately like the following code fragment.

```
        fquies = creat_filt(Quies_X_Dim,,Quies_Z_Dim,Qname);
        fmaneu = cr_filter(Maneu_X_Dim,Maneu_Z_Dim,Mname);
        p = create_plant(Maneu_X_Dim,,Pname);
        o = create_obs(Maneu_X_Dim, Maneu_Z_Dim,Oname);
        init_filter(fquies,p,o);
        current_f = fquies;
        current_time = 0.0;

        for(i=0;i<steps; i++)
        {
                current_time += timestep;
                delta_plant(p);
                measure(p,o);
                ekf(current_f,p,o);
                report(current_f,p);   /*simulation output*/
                if (current_f.norm_innov > thresh)
                        {
                        init_f2(fmaneu,current_f,p,o);
                        current_f = fmaneu;
                        }
        }
```

The basic loop can be embedded in an interactive, window-based environment or can simply accept keyboard input and output data to files.

6.1.2 Data Structures

```
extern struct Filter;

typedef struct Filter
        {
        int XDIM;        /*plant state dimension*/
        int ZDIM;        /*measurement dimension*/
        int name;        /*filter ID*/
        matrix_t H;      /*measurement matrix*/
        matrix_t F;      /*plant dynamics matrix  -- F and H may
not be used in extended filter...need
 f and h functions */
        matrix_t R;      /*measurement noise covariance*/
        matrix_t W;      /*Kalman gain matrix*/
        matrix_t S;      /*Innovation covariance*/
        matrix_t Q;      /*State noise covariance*/
        matrix_t P_est;  /*predicted state error covariance*/
        matrix_t xhat_est;      /*predicted state*/
        matrix_t rho;    /*fading memory likelihood function*/
        matrix_t norm_staterr;  /*norm. state err. for sims*/
        matrix_t norm_innov;    /*normalized innovation*/
        double alpha;           /*ratio for fading memory*/
        matrix_t (*compute_H)(); /* funcs for ekf matrices*/
        matrix_t (*compute_F)();
        matrix_t (*compute_R)();
        struct Filter  *next;    /*filter linking information*/
        struct Filter  *last;
        } *Filter_t;


typedef struct plant /* redesigned for each plant*/
        {
        int name;           /*plant ID*/
        int XDIM;           /*plant state dimension*/
        matrix_t F;         /*plant dynamics*/
        matrix_t x;         /*plant state*/
        matrix_t V;         /*plant noise vector*/
        matrix_t pure_x;    /*noiseless state*/
        double sqrtq;       /*noise standard deviation*/
        }  *plant_t;

typedef struct observer /* redesigned for each observer*/
        {
        int name;
        int OBSXDIM;
        int ZDIM;
        matrix_t obs_x; /*observed x*/
        matrix_t obs_x_pure; /*noiseless observed x*/
        double sqrtrx;
        double sqrtry;
        double sqrtrs; /*noise standard devs*/
        matrix_t w;
        matrix_t z;
        matrix_t pure_z; /*noiseless observation*/
        } *observer_t;
```

6.2 Computational Effort

Here we assess the computational requirements of the current implementation of the Kalman filter. Let X be the dimensionality of the state vector and Z that of the measurement vector.

Initializing a filter allocates memory for 34 matrices, varying in size from $X \times X$ to 1×1. At each iteration of the nonlinear (extended) Kalman filter, procedures must be run to evaluate $h(.)$ (the measurement equation), to predict the measurement, and to compute R (the measurement covariance), both of which can be functions of time. In the linear filter the H and R matrices are fixed during initialization.

During one iteration of the Kalman filter, as currently implemented, there are some 28 separate matrix operations (transpose, add, multiply, inverse). The primitive operation count for the matrix operations in the (linear) Kalman filter is as follows.

$$Multiplies = 2X^3 + 2Z^3 + 3X^2Z + 2XZ^2 + X^2 + Z^2 + 2XZ + Z$$

$$Adds = 2X^3 + 2Z^3 + 3X^2Z + 3XZ^2 + 3X^2 + 2Z^2 + 2XZ + 2Z + X$$

$$Copies = X + 3XZ + Z$$

If the ratio of execution times for multiplication to addition to copy is 5:1:0, $X = 4$, and $Z = 2$, each filter iteration costs the equivalent of 1918 additions. In that time one can perform the equivalent of a "5 x 5" Fast Fourier Transform (disregarding powers-of-two sizing). If $X = 10$, $Z = 5$, an equivalent of 27,370 additions is required, which is the effort for a "13 x 13" FFT. Nineteen steps of the larger Kalman filter or 273 steps of the smaller can run in the time needed for a 256 x 256 FFT.

The implementation above is the most straightforward (i.e. naive) one its author (Brown) could imagine. Numerically more robust methods, which can provide faster performance as well, include a whole range of "square root" methods. Gelb ([12], p. 106ff.) discusses them and also presents computer loading and timing analysis. Ron Daniel of the RRG has implemented ingenious methods for steady-state Kalman filtering on modern processor chips (M68030) that can run at 800 Hz.

Acknowledgements

This work is partially funded by the DARPA U.S. Army Engineering Topographic Laboratories Grant DACA76-85-C-0001 and the Air Force Systems Command (RADC, Griffiss AFB, NY) and Air Force OSR Contract F30602-85-C-0008, which supports the Northeast Artificial Intelligence Consortium. The College of Arts and Sciences of the University of Rochester provided sabbatical leave support for C. Brown. J. Leonard is supported by a Thouron fellowship. The assistance of Prof. J. Michael Brady and the RRG research and administrative staff is gratefully acknowledged.

7 REFERENCES

[1] Ahmed S. Abutaleb. Target image tracking using a nonlinear filter based on pontryagin minimum principle. *IEEE Transactions on Automatic Control*, AC-31(12):1170-1173, December 1986.

[2] Y. Bar-Shalom. Tracking methods in a multi-target environment. *IEEE Transactions on Automatic Control*, AC-23(4):618-626, August 1978.

[3] Y. Bar-Shalom and T.E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.

[4] C. M. Brown. Gaze controls with interactions and delays. In *DARPA IU Workshop. Submitted IEEE-TSMC*, 1989.

[5] C. M. Brown. Prediction in gaze and saccade control. *In preparation*, April 1989.

[6] C. M. Brown. *The Rochester Robot*. Technical Report 257, University of Rochester, September 1988.

[7] C.Y. Chong. Hierarchical estimation. In *2nd MIT/ONR CCC Workshop*, Monterey, CA., 1979.

[8] K.P.Dunn D. Willner, C.B.Chang. Kalman filter algorithm for a multi-sensor system. In *15th IEEE Conf. Decision Control*, Clearwater, Florida, 1976.

[9] H.F. Durrant-Whyte. Uncertain geometry in robotics. *IEEE J. Robotics and Automation*, 4(1):23-31, 1988.

[10] H.W. Sorenson - Guest Ed. Special issue on applications of kalman filtering. *IEEE Transactions on Automatic Control*, AC-28(3), March 1983.

[11] Anita M. Flynn. Combining sonar and infrared sensors for mobile robot navigation. *International Journal of Robotics Research*, 6(3), 1988. In the press.

[12] Arthur C. Gelb. *Applied Optimal Estimation*. The MIT Press, 1974.

[13] G. Hager and H.F. Durrant-Whyte. Information and multi-sensor coordination. In *Uncertainty in Artificial Intelligence*, pages 381–394, North Holland, 1988.

[14] John C. T. Hallam. *Intelligent Automatic Interpretation of Active Marine Sonar*. PhD thesis, University of Edinburgh, 1984.

[15] C.J. Harris and I. White. *Advances in Command, Control and Communication Sytems*. IEE Press, 1987.

[16] H.R. Hashemipour, S. Roy, and A.J. Laub. Decentralized structures for parallel kalman filtering. *IEEE Trans. Automatic Control*, 33(1):88–93, 1988.

[17] T. Henderson. *Workshop on Multi-Sensor Integration*. Technical Report UUCS-87-006, U. Utah Computer Science, 1987.

[18] Richard J. Kenefic. Optimum tracking of a maneuvering target in clutter. *IEEE Transactions on Automatic Control*, AC-26(3), June 1981.

[19] Robert M. Kuczewski. Neural network approaches to multi-target tracking. In *IEEE First International Conference of Neural Networks*, June 1987.

[20] J. Leonard. Active sensor control for mobile robotics. 1988. First Year Report, RRG, U. Oxford.

[21] Alan McIvor. Edge detection in dynamic vision. In *AVC88: Proceedings of the Fourth Alvey Vision Conference*, August-September 1988.

[22] C. L. Morefield. Application of 0-1 integer programming to multitarget tracking problems. *IEEE Transactions on Automatic Control*, AC-22(6), June 1977.

[23] Shozo Mori, C. Chong, E. Tse, and R. Wishner. Tracking and classifying multiple targets without a priori identification. *IEEE Transactions on Automatic Control*, AC-31(5), May 1986.

[24] B. Rao and H. Durrant-Whyte. A fully decentralized algorithm for multi-sensor kalman filtering. In *28th IEEE Conf. on Decision and Control*, December 1989.

[25] Donald B. Reid. An algorithm for tracking multple targets. *IEEE Transactions on Automatic Control*, AC-24(6), December 1979.

[26] Guy L. Scott. Schroedinger waves in tracking and correspondence applications. In *IEEE CVPR89*, August 1989.